

Database Semantics

Robert E. Kent

Ontologos

Abstract. This paper, the first step to connect relational databases with systems consequence (Kent [5]), is concerned with the semantics of relational databases. It aims to study system consequence in the logical/semantic system of relational databases. The paper, which was inspired by and which extends a recent set of papers on the theory of relational database systems (Spivak [6] [7]), is linked with work on the Information Flow Framework (IFF [9]) connected with the ontology standards effort (SUO), since relational databases naturally embed into first order logic. The database semantics discussed here is concerned with the conceptual level of database architecture. We offer both an intuitive and technical discussion. Corresponding to the notions of primary and foreign keys, relational database semantics takes two forms: a distinguished form where entities are distinguished from relations, and a unified form where relations and entities coincide. The distinguished form corresponds to the theory presented in (Spivak [6]). The unified form, a special case of the distinguished form, corresponds to the theory presented in (Spivak [7]). A later paper will discuss various formalisms of relational databases, such as relational algebra and first order logic, and will complete the description of the relational database logical environment.

Keywords: database systems, database schemas, relational tables, primary and foreign keys, morphisms of databases, relational algebra, first order logic, system consequence.

1 Introduction

The author's "Systems Consequence" paper (Kent [5]) is a very general theory and methodology for specification and inter-operation of systems of information resources. The generality comes from the fact that it is independent of the logical/semantic system (institution) being used. This is a wide-ranging theory, based upon ideas from information flow (Barwise and Seligman [1]), formal concept analysis (Wille and Ganter et al [2]), the theory of institutions (Goguen et al [3]), and the lattice of theories notion (Sowa), for the integration of both formal and semantic systems independent of logical environment. In order to better understand the motivations of that paper and to be able more readily to apply its concepts, in the future it will be important to study system consequence in various particular logical/semantic systems. This paper aims to do just that for the logical/semantic system of relational databases. The paper, which

was inspired by and which extends a recent set of papers on the theory of relational database systems (Spivak [6],[7]), is linked with work on the Information Flow Framework (IFF [9]) connected with the ontology standards effort (SUO), since relational databases naturally embed into first order logic. We offer both an intuitive and technical discussion. Corresponding to the notions of primary and foreign keys, relational database semantics takes two forms: a distinguished form where entities are distinguished from relations, and a unified form where relations and entities coincide. The distinguished form corresponds to the theory presented in the paper (Spivak [6]). We extend Spivak's treatment of tables from the static case of a single entity classification (type specification) to the dynamic case of classifications varying along infomorphisms. Our treatment of relational databases as diagrams of tables differs from Spivak's sheaf theory of databases. The unified form, a special case of the distinguished form, corresponds to the theory presented in the paper (Spivak [7]). The unified form has a graphical presentation, which corresponds to the sketch theory of databases (Johnson and Rosebrugh [4]) and the resource description framework (RDF). This paper, which is the first step to connect relational databases with systems consequence, is concerned with the semantics of relational databases. A later paper will discuss various formalisms of relational databases, such as relational algebra and first order logic. Section 2 discusses the relational data model. Section 3 describes our representation for the table concept, both defining a category of tables, and proving that this category is complete (joins exist) and cocomplete (unions exist). Section 4 represents the relational database concept as a diagram of tables linked by the generalization-specialization of projections. Morphisms of relational databases are defined. Canonical examples of both are discussed. Finally, section 5 summarizes the results and gives some concluding remarks.

2 Relational Data Model

The paper defines an architectural semantics for the relational data model.¹ All information in the relational model is represented within relations. A relational database is a collection of relations (relational tables, or just tables). A table is represented as an array, organized into rows and columns. The rows are called the tuples (records) of the table, whereas the columns are called the attributes of the table. Both rows (tuples) and columns are unordered. In the basic relational data model all the components can be resolved into sets and functions.²

The basic relational building block is the data domain represented by an entity type $x \in X$, where X is the type set of an entity classification $\mathcal{E} = \langle X, Y, \models_{\mathcal{E}} \rangle$, whose instance set is a universe of data values Y local to the database. An entity instance $y \in Y$ is classified by an entity type $x \in X$ when $y \models_{\mathcal{E}} x$. Within the classification \mathcal{E} the entity type $x \in X$ represents its extent, which is

¹ Older architectures of data include the hierarchical model and network model. Of these, nothing will be said. A newer architecture of data, called the object-relation-object model, is a presentation form for the relational data model described here.

² The basic relational data model is defined on the category **Set** of sets and functions.

the domain of data values $\mathbf{ext}_{\mathcal{E}}(x) = \{y \in Y \mid y \models_{\mathcal{E}} x\}$. We extend the classification to generalized elements. An indexed collection of entity types $\{(i, s_i) \mid i \in I, s_i \in X\}$ is called an \mathcal{E} -signature. It is denoted by the pair $\langle I, s \rangle$ and represented as a map $I \xrightarrow{s} X$ from index set to entity type set. An indexed collection of entity instances $\{(j, t_j) \mid j \in J, t_j \in Y\}$ is called an \mathcal{E} -tuple. A tuple represents an object; either a concrete, physical object or an abstract, conceptual object. It is denoted by the pair $\langle J, t \rangle$ and represented as a map $J \xrightarrow{t} Y$ from index set to the universe. The indexing set is called the arity of the signature or tuple. A \mathcal{E} -tuple $\langle J, t \rangle$ is classified by an \mathcal{E} -signature $\langle I, s \rangle$, denote by $t \models_{\mathcal{E}} s$, when they have the same arity $J = I$ and enjoy pointwise classification $t_i \models_{\mathcal{E}} s_i$ for all $i \in I$. The extent of an \mathcal{E} -signature $\langle I, s \rangle$ is its tuple set $\mathbf{tup}_{\mathcal{E}}(I, s) = \{t \mid t \models_{\mathcal{E}} s\}$.

Let \mathcal{T} be a relational table in a database based on the entity classification \mathcal{E} . An attribute of \mathcal{T} is an ordered pair (i, s_i) consisting of an attribute name $i \in I$ and an entity type $s_i \in X$, where I is the arity of the table. The collection of attributes of \mathcal{T} forms its schema $\langle I, s, X \rangle$, where $\langle I, s \rangle$ is an \mathcal{E} -signature. A tuple of \mathcal{T} is an \mathcal{E} -tuple that is classified by the table signature $\langle I, s \rangle$. Hence, the tuple set of \mathcal{T} is the set $\mathbf{tup}_{\mathcal{E}}(I, s)$. Each tuple of \mathcal{T} must be uniquely identifiable by some combination (one or more) of its attribute values. This combination is referred to as the primary key. Without loss of generality, we assume that (primary) keys are single attributes. In addition, we conceptually separate the primary key attribute from the rest of the table and use it for indexing. Hence, the table \mathcal{T} is an indexed collection of \mathcal{E} -tuples $\mathcal{T} = \{(k, \tau_k) \mid \tau_k \in \mathbf{tup}_{\mathcal{E}}(I, s), k \in K\}$, where K is the set of primary keys of the table; that is, the table is represented as a map $K \xrightarrow{\tau} \mathbf{tup}_{\mathcal{E}}(I, s)$ from keys to tuples.

Here is an small example of a relational database for a company in unified form, which illustrates both primary keys (\blacktriangle) and foreign keys (\triangle). It contains two relational tables, an employee table **Emp** and a department table **Dept**, which are indexed by primary keys and linked by foreign keys.

<i>emp</i> : Emp	<i>name</i> : Str	<i>addr</i> : Str	<i>dept</i> : Dept
e1	Plato	Greece	d1
e2	Aquinas	Italy	d2
e3	Decartes	France	d1

\blacktriangle \triangle

<i>dept</i> : Dept	<i>name</i> : Str	<i>mngtr</i> : Emp
d1	Sales	e3
d2	Production	e2

\blacktriangle \triangle

In this example, the entity (relation) types are **Dept**, **Emp** and **Str**. In the employee relational table **Emp**, the arity is $\{name, addr, dept\}$, the signature is $\{(name, \mathbf{Str}), (addr, \mathbf{Str}), (dept, \mathbf{Dept})\}$, and the (primary) key set is $\{e1, e2, e3\}$. Dotted items indicate relations (types or instances) being used as entities, since this is in unified form.

In the relational data model, there are three inherent integrity constraints: entity integrity, domain integrity, and referential integrity. Entity integrity asserts that every table must have a primary key column in which each entry identifies its own row (tuple). Domain integrity asserts that each data entry in a column must be of the type of that column. Entity and domain integrity are requirements for the distinguished form of database semantics. Entity integrity says there must be a tuple function from the set of (primary) keys, and domain

integrity says that image tuples must be classified by the table signature $\langle I, s \rangle$. Hence, entity and domain integrity assert the existence of the tuple or content function $t : K \rightarrow \mathbf{tup}_{\mathcal{E}}(I, s)$. Referential integrity asserts that each entry in a foreign key column of a referencing table must occur in the primary key column of the referenced table. Referential integrity is a requirement for the unified form of database semantics. Referential integrity says there must be a function from a foreign key column of a referencing table to the primary key column of the referenced table. Hence, referential integrity asserts the existence of the functions in the sketch interpretation of a relational database.

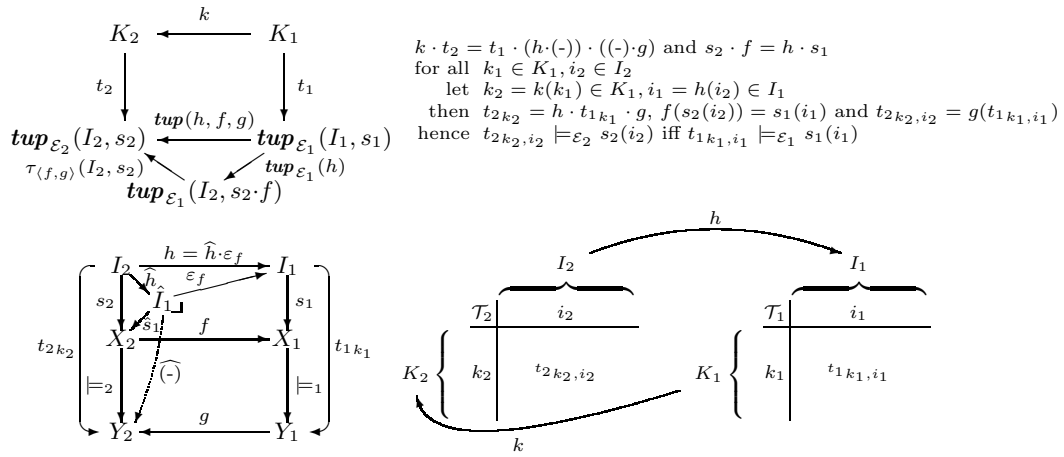
The information in a database is accessed by specifying queries, which use operations such as select to identify tuples, project to identify attributes, and join to combine tables. In this paper, projection refers to a primitive generalization-specialization operation between pairs of relational tables (they are specified by the database schema, project from joined table to components, or other), whereas join is a composite operation on a linked collection of tables. Selection is a special case of join, which uses reference relations (tables).

3 Tables

A table (database relation) $\mathcal{T} = \langle \mathcal{S}, \mathcal{E}, K, t \rangle$ has an underlying (simple) schema $\mathcal{S} = \langle X, I, s \rangle$ with a set of entity types X and an X -signature $\langle I, s \rangle \in (\mathbf{Set} \downarrow X)$, an entity classification $\mathcal{E} = \langle X, Y, \models_{\mathcal{E}} \rangle$ with a common (entity) type set component $X \in \mathbf{Set}$ and a local universe of entity instances $Y \in \mathbf{Set}$, a set K of (primary) keys, and a tuple function $t : K \rightarrow \mathbf{tup}_{\mathcal{E}}(I, s)$ mapping keys to \mathcal{E} -tuples of type (signature) $\langle I, s \rangle$. Equivalently, it is an object in the comma category of \mathcal{E} -tables $\mathcal{T} \in (\mathbf{Set} \downarrow \mathbf{tup}_{\mathcal{E}})$.

A table morphism (morphism of database relations) $\langle h, f, g, k \rangle : \mathcal{T}_1 = \langle \mathcal{S}_1, \mathcal{E}_1, K_1, t_1 \rangle \rightarrow \langle \mathcal{S}_2, \mathcal{E}_2, K_2, t_2 \rangle = \mathcal{T}_2$ consists of a (simple) schema morphism $\langle h, f \rangle : \mathcal{S}_2 = \langle X_2, I_2, s_2 \rangle \rightarrow \langle X_1, I_1, s_1 \rangle = \mathcal{S}_1$ with a function on entity types $f : X_2 \rightarrow X_1$ and an X_1 -signature morphism $h : \Sigma_f(I_2, s_2) = \langle I_2, s_2 \cdot f \rangle \rightarrow \langle I_1, s_1 \rangle$, an entity infomorphism $\langle f, g \rangle : \mathcal{E}_2 = \langle X_2, Y_2, \models_{\mathcal{E}_2} \rangle \rightrightarrows \langle X_1, Y_1, \models_{\mathcal{E}_1} \rangle = \mathcal{E}_1$ with a common (entity) type function component $f : X_2 \rightarrow X_1$ and a universe (entity instance) function $g : X_1 \rightarrow X_2$, and a key function $k : K_1 \rightarrow K_2$, which satisfy the condition $k \cdot t_2 = t_1 \cdot \mathbf{tup}(h, f, g)$, where $\mathbf{tup}(h, f, g) \doteq \mathbf{tup}_{\mathcal{E}_1}(h) \cdot \tau_{\langle f, g \rangle}(I_2, s_2) = (h \cdot (-)) \cdot ((-) \cdot g) : \mathbf{tup}_{\mathcal{E}_1}(I_1, s_1) \rightarrow \mathbf{tup}_{\mathcal{E}_2}(I_2, s_2)$.³ Table morphisms are illustrated in Figure 1. Here we see that table morphisms have the pleasing property that corresponding entries in the source and target tables satisfy the infomorphism condition from the theory of information flow (Barwise and Seligman [1]). Composition of morphisms is defined component-wise. Let \mathbf{Tbl} denote the category of tables (database relations) with the two projections (*sch* is called the schema functor) $\mathbf{Sch} \xleftarrow{\text{sch}} \mathbf{Tbl}^{\text{op}} \xrightarrow{\text{cls}} \mathbf{Cls}$ and the key functor $\mathbf{Tbl} \xrightarrow{\text{key}} \mathbf{Set}$.

³ Since the table tuple function embodies the entity/domain integrity constraints (Section 2), this condition on morphisms asserts the preservation of data integrity.



This four-part figure illustrates the condition on table morphisms. It has been annotated to help guide the understanding. The condition is symbolically stated in terms of set functions in the line of text just above. The top left diagram illustrates the condition, and the bottom left diagram expands on this. The top right diagram text is more detailed in terms of a source key $k_1 \in K_1$. Here we see appearance of the infomorphism condition

$$g(t_{1k_1, i_1}) \models_{\mathcal{E}_2} s_2(i_2) \text{ iff } t_{1k_1, i_1} \models_{\mathcal{E}_1} f(s_2(i_2)).$$

Finally, the bottom left figure illustrates the effect of the morphism on source/target tables \mathcal{T}_1 and \mathcal{T}_2 .

Fig. 1. Table Morphism

$$\begin{array}{ccccc}
& \langle S_2, \mathcal{E}_2, K_2, t_2 \rangle & \xleftarrow{\langle h, f, g, k \rangle} & \langle S_1, \mathcal{E}_1, K_1, t_1 \rangle & \\
K_2 \xleftarrow{k} K_1 & \mathbf{Set}^{\text{op}} & \xleftarrow{\text{key}^{\text{op}}} & \mathbf{Tbl}^{\text{op}} & \xrightarrow{\text{cls}} & \mathbf{Cls} & \langle X_2, Y_2, \models_{\mathcal{E}_2} \rangle \xrightarrow{\langle f, g \rangle} \langle X_1, Y_1, \models_{\mathcal{E}_1} \rangle \\
& \downarrow & & \downarrow \text{sch} & \searrow \text{Sdsgn} & \downarrow \text{typ} \\
& \mathbf{1} & \xleftarrow{\quad} & \mathbf{Sch} & \xrightarrow{\text{typ}} & \mathbf{Set} & X_2 \xrightarrow{f} X_1 \\
& & & \langle X_2, I_2, s_2 \rangle & \xrightarrow{\langle h, f \rangle} & \langle X_1, I_1, s_1 \rangle
\end{array}$$

We can have three indexing categories for tables: classifications, schema or semidesignations. Each has their uses: classification indexing proves that the category of tables is complete (and the fibers help explain database fibers), semidesignation indexing proves that the category of tables is cocomplete, and schema indexing follows the true formal-semantics distinction.

3.1 Classification Indexed Category.

For any fixed classification \mathcal{E} , the \mathcal{E}^{th} -fiber category with respect to the classification functor $\mathbf{Tbl}^{\text{op}} \xrightarrow{\text{cls}} \mathbf{Cls}$, called the category of \mathcal{E} -tables, is the comma category associated with the tuple functor $\mathbf{tup}_{\mathcal{E}} : (\mathbf{Set} \downarrow X)^{\text{op}} \rightarrow \mathbf{Set}$.

$$\mathbf{Set} \xleftarrow{\text{key}_{\mathcal{E}}} \mathbf{Tbl}(\mathcal{E}) = (\mathbf{Set} \downarrow \mathbf{tup}_{\mathcal{E}}) \xrightarrow{\text{sign}_{\mathcal{E}}^{\text{op}}} (\mathbf{Set} \downarrow X)^{\text{op}}.$$

It has key and signature projection functors, an equivalent natural transformation $\tau : \text{key}_{\mathcal{E}} \Rightarrow \text{sign}_{\mathcal{E}}^{\text{op}} \circ \mathbf{tup}_{\mathcal{E}}$, and is described as follows. A fiber object $\mathcal{T} \in \mathbf{Tbl}(\mathcal{E})$, or an \mathcal{E} -table (database \mathcal{E} -relation), is any table $\mathcal{T} \in \mathbf{Tbl}$ with entity classification $\text{cls}(\mathcal{T}) = \mathcal{E}$ and tuple (content) function $t : K \rightarrow \mathbf{tup}_{\mathcal{E}}(I, s)$ mapping each key (abstract tuple) to a (concrete) \mathcal{E} -tuple in the extent of $\langle I, s \rangle$. A fiber morphism in $\mathbf{Tbl}(\mathcal{E})$ is any table morphism $\langle h, k \rangle : \mathcal{T} = \langle \mathcal{S}, \mathcal{E}, K, t \rangle \leftarrow \langle \tilde{\mathcal{S}}, \mathcal{E}, \tilde{K}, \tilde{t} \rangle = \tilde{\mathcal{T}}$ in \mathbf{Tbl} with identity infomorphism $\text{id}_{\mathcal{E}} = \langle \text{id}_X, \text{id}_Y \rangle$. It consists of a signature morphism $h : \langle I, s \rangle \rightarrow \langle \tilde{I}, \tilde{s} \rangle$ and a key function $k : \tilde{K} \rightarrow K$, which satisfy the condition $k \cdot t = \tilde{t} \cdot \mathbf{tup}_{\mathcal{E}}(h)$.

Proposition 1. *There is an indexed category of tables $\mathbf{tbl} : \mathbf{Cls}^{\text{op}} \rightarrow \mathbf{Cat}$ from (the opposite of) the category of classifications and infomorphisms to the category of categories and functors.⁴ The (opposite of the) fibered category corresponding to this (its Grothendieck construction) is isomorphic to the category of tables with the classification functor projection $\mathbf{Tbl}^{\text{op}} \xrightarrow{\text{cls}} \mathbf{Cls}$.*

Proposition 2. *The category of \mathcal{E} -tables $\mathbf{Tbl}(\mathcal{E}) = (\mathbf{Set} \downarrow \mathbf{tup}_{\mathcal{E}})$ is complete, its key projection functor $\mathbf{Tbl}(\mathcal{E}) \xrightarrow{\text{key}_{\mathcal{E}}} \mathbf{Set}$ is continuous and its signature projection functor $\mathbf{Tbl}(\mathcal{E})^{\text{op}} \xrightarrow{\text{sign}_{\mathcal{E}}} (\mathbf{Set} \downarrow X)$ is cocontinuous.*

⁴ The table indexing functor is the composition $\mathbf{tbl} = \mathbf{tup}^{\text{op}} \circ \mathbf{comma} \circ (-)^{\text{op}} : \mathbf{Cls}^{\text{op}} \rightarrow (\mathbf{Adj} \downarrow \mathbf{Set})^{\text{op}} \rightarrow \mathbf{Cat} \rightarrow \mathbf{Cat}$ of a tuple functor $\mathbf{Cls} \xrightarrow{\text{tup}} (\mathbf{Adj} \downarrow \mathbf{Set})$ and a comma category functor $(\mathbf{Adj} \downarrow \mathbf{Set})^{\text{op}} \xrightarrow{\text{comma}} \mathbf{Cat}$.

Proof. (We have both an abstract and a useful concrete proof, but only have room for the former.) The category $(\mathbf{Set} \downarrow X)^{\text{op}}$ is complete, since $(\mathbf{Set} \downarrow X)$ is cocomplete. The tuple functor $\mathbf{tup}_{\mathcal{E}}$ is continuous.⁵

The category of \mathcal{E} -tables $\mathbf{Tbl}(\mathcal{E})$ is the semantical domain for a relational database \mathcal{D} with entity classification \mathcal{E} . Completeness of $\mathbf{Tbl}(\mathcal{E})$ means that, not just binary, but database joins over arbitrary diagrams of tables of \mathcal{D} can be computed.

Proposition 3. *For any infomorphism $\langle f, g \rangle : \mathcal{E}_2 \rightarrow \mathcal{E}_1$, the table fiber functor $\mathbf{tbl}_{\langle f, g \rangle} : \mathbf{Tbl}(\mathcal{E}_1) \rightarrow \mathbf{Tbl}(\mathcal{E}_2)$ is continuous.*

Continuity of $\mathbf{tbl}_{\langle f, g \rangle}$ means that database joins are preserved: database joins of \mathcal{E}_1 -tables are mapped to database joins of \mathcal{E}_2 -tables.

Theorem 1. *The category of tables \mathbf{Tbl} is complete.*

Proof. The indexing category \mathbf{Cls} is complete, the fiber category $\mathbf{tbl}(\mathcal{E})$ is complete for each classification \mathcal{E} , and the fiber functor $\mathbf{tbl}_{\langle f, g \rangle} : \mathbf{tbl}(\mathcal{E}_1) \rightarrow \mathbf{tbl}(\mathcal{E}_2)$ is continuous for each infomorphism $\langle f, g \rangle : \mathcal{E}_2 \rightarrow \mathcal{E}_1$. Hence, this is an application of a theorem of Tarlecki, Burstall and Goguen [8].⁶

3.2 Schema Indexed Category.

For any fixed (simple) schema $\mathcal{S} = \langle X, I, s \rangle$, the \mathcal{S}^{th} -fiber category $\mathbf{Tbl}(\mathcal{S})$ with respect to the schema functor $\mathbf{Tbl}^{\text{op}} \xrightarrow{\text{sch}} \mathbf{Sch}$, called the category of \mathcal{S} -tables, is the comma category with key and X -classification projection functors⁷

$$\mathbf{Set} \xleftarrow{\text{key}_{\mathcal{S}}} \mathbf{Tbl}(\mathcal{S}) = (\mathbf{Set} \downarrow \mathbf{tup}_{\mathcal{S}}) \xrightarrow{\text{cls}_{\mathcal{S}}^{\text{op}}} \mathbf{Cls}(X)^{\text{op}}.$$

It is described as follows. A fiber object $\mathcal{T} \in \mathbf{Tbl}(\mathcal{S})$, or an \mathcal{S} -table (database \mathcal{S} -relation), is any table $\mathcal{T} \in \mathbf{Tbl}$ with (simple) schema $\text{sch}(\mathcal{T}) = \mathcal{S}$. A fiber morphism in $\mathbf{Tbl}(\mathcal{S})$ is any table morphism $\langle g, k \rangle : \mathcal{T} = \langle \mathcal{S}, \mathcal{E}, K, t \rangle \leftarrow \langle \mathcal{S}, \tilde{\mathcal{E}}, \tilde{K}, \tilde{t} \rangle = \tilde{\mathcal{T}}$ in \mathbf{Tbl} with identity (simple) schema morphism $\text{id}_{\mathcal{S}} = \langle \text{id}_X, \text{id}_I \rangle$. It consists of a universe (entity instance) function $g : \tilde{Y} \rightarrow Y$ defining an entity infomorphism $\langle I_X, g \rangle : \mathcal{E} = \langle X, Y, \models_{\mathcal{E}} \rangle \rightrightarrows \langle X, \tilde{Y}, \tilde{\models} \rangle = g^{-1}(\mathcal{E}) = \tilde{\mathcal{E}}$ and hence

⁵ If \mathcal{E} and \mathcal{B} are complete, and both $T : \mathcal{E} \rightarrow \mathcal{C}$ and $S : \mathcal{B} \rightarrow \mathcal{C}$ are continuous functors, then the comma category $(T \downarrow S)$ is also complete and the projection functors $(T \downarrow S) \rightarrow \mathcal{E}$ and $(T \downarrow S) \rightarrow \mathcal{B}$ are limit preserving.

⁶ If $\mathbf{C} : \mathbf{I}^{\text{op}} \rightarrow \mathbf{Cat}$ is an indexed category such that \mathbf{I} is complete, \mathbf{C}_i is complete for all indices $i \in \mathbf{I}$, and $\mathbf{C}_{\sigma} : \mathbf{C}_j \rightarrow \mathbf{C}_i$ is continuous for all index morphisms $\sigma : i \rightarrow j$, then $\mathbf{Gr}(\mathbf{C})$ is complete.

⁷ The tuple functor $\mathbf{tup}_{\mathcal{S}} : \mathbf{Cls}(X)^{\text{op}} \rightarrow \mathbf{Set}$ maps an X -classification $\mathcal{E} = \langle X, Y, \models \rangle$ to the tuple set $\mathbf{tup}_{\mathcal{S}}(Y, \models) = \mathbf{tup}_{\mathcal{E}}(I, S)$ and maps an X -infomorphism $\langle I_X, g \rangle : \mathcal{E}_2 = \langle X, Y_2, \models_2 \rangle \rightrightarrows \langle X, Y_1, \models_1 \rangle = g^{-1}(\mathcal{E}_2) = \mathcal{E}_1$ with instance function $g : Y_1 \rightarrow Y_2$ to the tuple function $\mathbf{tup}_{\mathcal{S}}(g) = \tau_{\langle I_X, g \rangle}(I, s) = (-) \cdot g : \mathbf{tup}_{\mathcal{S}}(Y_1, \models_1) = \mathbf{tup}_{\mathcal{E}_1}(I, S) \rightarrow \mathbf{tup}_{\mathcal{E}_2}(I, S) = \mathbf{tup}_{\mathcal{S}}(Y_2, \models_2)$.

the presheaf morphism $\langle (\mathbf{Set} \downarrow X), \mathbf{tup}_{\mathcal{E}} \rangle \xrightarrow{\langle 1, \tau_{\langle 1X, g \rangle} \rangle} \langle (\mathbf{Set} \downarrow X), \mathbf{tup}_{\tilde{\mathcal{E}}} \rangle$ with tuple natural transformation $\tau_{\langle 1X, g \rangle} : \mathbf{tup}_{\tilde{\mathcal{E}}} \Rightarrow \mathbf{tup}_{\mathcal{E}}$, and a key function $k : \tilde{K} \rightarrow K$, which satisfy the condition $k \cdot t = \tilde{t} \cdot \mathbf{tup}(g)$.⁸

Proposition 4. *There is an indexed category of tables $\mathbf{tbl} : \mathbf{Sch}^{\text{op}} \rightarrow \mathbf{Cat}$, whose Grothendieck construction (fibered category) is (the opposite of) the category of tables with the schema functor projection $\mathbf{Tbl}^{\text{op}} \xrightarrow{\text{sch}} \mathbf{Sch}$.*

3.3 Semidesignation Indexed Category.

A semidesignation $\mathcal{S} = \langle I, s, \mathcal{E} \rangle$, consists of a schema $\langle X, I, s \rangle$, and an entity classification $\mathcal{E} = \langle X, Y, \models_{\mathcal{E}} \rangle$ with a common (entity) type set component X . A semidesignation morphism $\langle h, f, g \rangle : \mathcal{S}_2 \rightarrow \mathcal{S}_1$ consists of a schema morphism $\langle h, f \rangle : \mathcal{S}_2 \rightarrow \mathcal{S}_1$ and an entity infomorphism $\langle f, g \rangle : \mathcal{E}_2 = \langle X_2, Y_2, \models_{\mathcal{E}_2} \rangle \rightrightarrows \langle X_1, Y_1, \models_{\mathcal{E}_1} \rangle = \mathcal{E}_1$ with a common (entity) type function component $f : X_2 \rightarrow X_1$. For any semidesignation $\mathcal{S} = \langle I, s, \mathcal{E} \rangle$, the set of tuples of \mathcal{S} is $\mathbf{tup}(\mathcal{S}) = \mathbf{tup}_{\mathcal{E}}(I, s)$, the set of \mathcal{E} -tuples in the extent of $\langle I, s \rangle$.

Lemma 1. *Any semidesignation morphism $\langle h, f, g \rangle : \mathcal{S}_2 \rightarrow \mathcal{S}_1$ defines a tuple function $\mathbf{tup}(h, f, g) : \mathbf{tup}(\mathcal{S}_1) = \mathbf{tup}_{\mathcal{E}_1}(I_1, s_1) \rightarrow \mathbf{tup}_{\mathcal{E}_2}(I_2, s_2) = \mathbf{tup}(\mathcal{S}_2)$. Hence, there is a tuple functor $\mathbf{tup} : \mathbf{Sdsign}^{\text{op}} \rightarrow \mathbf{Set}$.*

Proposition 5. *The category of tables is the comma category*

$$\mathbf{Set} \xleftarrow{\text{key}} \mathbf{Tbl} = (\mathbf{Set} \downarrow \mathbf{tup}) \xrightarrow{\text{sdsign}} \mathbf{Sdsign}^{\text{op}} \xrightarrow{\text{cls}^{\text{op}}} \mathbf{Cls}^{\text{op}}$$

associated with the tuple functor $\mathbf{tup} : \mathbf{Sdsign}^{\text{op}} \rightarrow \mathbf{Set}$. The category of tables is cocomplete.

Proof. The opposite category of semidesignations $\mathbf{Sdsign}^{\text{op}}$ is cocomplete, since \mathbf{Sdsign} is complete.⁹

4 Relational Databases

A relational database $\mathcal{D} = \langle \mathcal{S}, \mathcal{E}, K, \tau \rangle$ is a naturally connected diagram of tables. It has an underlying relational database schema $\mathcal{S} = \langle \mathbf{R}, X, \mathbf{S} \rangle$ ¹⁰ with

⁸ The components determining variance between $\mathcal{T} = \langle \mathcal{S}, \mathcal{E}, K, t \rangle$ and $\tilde{\mathcal{T}} = \langle \mathcal{S}, \tilde{\mathcal{E}}, \tilde{K}, \tilde{t} \rangle$ are the entity instance function (varying instances and their incidence or classification relations) and the key function (varying the set of keys and the tuple natural transformations).

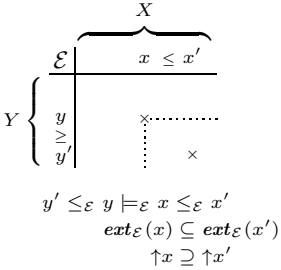
⁹ If \mathcal{A} and \mathcal{B} are cocomplete, $T : \mathcal{A} \rightarrow \mathcal{C}$ is a cocontinuous functor, and $S : \mathcal{B} \rightarrow \mathcal{C}$ is any functor (not necessarily cocontinuous), then the comma category $(T \downarrow S)$ will also be cocomplete.

¹⁰ A relational database schema $\mathcal{S} = \langle \mathbf{R}, X, \mathbf{S} \rangle$ consists of a category of relation symbols \mathbf{R} , a set of entity types X , and a signature functor $\mathbf{S} : \mathbf{R} \rightarrow (\mathbf{Set} \downarrow X)$. Any rela-

a category of relation types (symbols) \mathbf{R} linked by generalization-specialization, a set of entity types X , and a signature functor $\mathbf{S} : \mathbf{R} \rightarrow (\mathbf{Set} \downarrow X)$, an entity classification $\mathcal{E} = \langle X, Y, \models_{\mathcal{E}} \rangle$ with a common (entity) type set component X and a local universe of entity instances Y , a key functor $\mathbf{K} : \mathbf{R}^{\text{op}} \rightarrow \mathbf{Set}$, and a tuple natural transformation $\tau : \mathbf{K} \Rightarrow \mathbf{S}^{\text{op}} \circ \mathbf{tup}_{\mathcal{E}}$. Equivalently, it consists of a table functor $\mathbf{T} : \mathbf{R}^{\text{op}} \rightarrow (\mathbf{Set} \downarrow \mathbf{tup}_{\mathcal{E}})$, where $\tau = \mathbf{T}\tau_{\mathcal{E}}$ and $\tau_{\mathcal{E}} : \mathbf{key}_{\mathcal{E}} \Rightarrow \mathbf{sign}_{\mathcal{E}}^{\text{op}} \circ \mathbf{tup}_{\mathcal{E}}$ is the tuple natural transformation that is an integral component of the comma category $(\mathbf{Set} \downarrow \mathbf{tup}_{\mathcal{E}})$. Here are some examples of relational databases.

Table. A table (database relation) $\mathcal{T} = \langle \mathcal{S}, \mathcal{E}, K, t \rangle$ with entity classification $\mathcal{E} = \langle X, Y, \models_{\mathcal{E}} \rangle$, schema $\mathcal{S} = \langle X, I, s \rangle$, tuple set K , and tuple function $t : K \rightarrow \mathbf{tup}_{\mathcal{E}}(I, s)$, is a one-object relational database with the same entity classification, the terminal category of relation types (symbols) $\mathbf{1} = \{*\}$, the signature functor with single X -signature $\mathbf{1} \xrightarrow{\langle I, s \rangle} (\mathbf{Set} \downarrow X)$, the key functor with single key set $\mathbf{1}^{\text{op}} = \mathbf{1} \xrightarrow{K} \mathbf{Set}$, the tuple natural transformation with single component tuple function $t : K \rightarrow \mathbf{tup}_{\mathcal{E}}(I, s)$, and the table functor with single \mathcal{E} -table $\mathbf{1}^{\text{op}} = \mathbf{1} \xrightarrow{\mathcal{T}} (\mathbf{Set} \downarrow \mathbf{tup}_{\mathcal{E}})$.

Classification. A classification $\mathcal{E} = \langle X, Y, \models_{\mathcal{E}} \rangle$ is a relational database $\mathbf{db}(\mathcal{E}) = \langle \geq_{\mathcal{E}}, \uparrow_{\mathcal{E}}, \mathcal{E}, \mathbf{ext}_{\mathcal{E}}, \tau_{\mathcal{E}} \rangle$, where the entity classification is itself ($\mathbf{db} \circ \mathbf{cls} = \mathbf{id}_{\mathbf{Cls}}$).¹¹ The additional components are described as follows. The category of relation types (symbols) is the reverse conceptual preorder (generalization-specialization order) on entity types $\langle X, \geq_{\mathcal{E}} \rangle$ with $x' \geq_{\mathcal{E}} x$ when $\mathbf{ext}_{\mathcal{E}}(x') \supseteq \mathbf{ext}_{\mathcal{E}}(x)$; that is, when x' is at least as general as x . The signature functor is the principle filter operator $\langle X, \geq_{\mathcal{E}} \rangle \xrightarrow{\uparrow_{\mathcal{E}}} (\mathbf{Set} \downarrow X)$; it maps an entity type $x \in X$ to the X -signature $\langle \uparrow x, \mathbf{inc}_x \rangle$, where $\uparrow x = \{x' \in X \mid x' \geq_{\mathcal{E}} x\}$ is the principle filter of x and $\mathbf{inc}_x : \uparrow x \hookrightarrow X$ is inclusion; and it maps an ordering $x' \geq_{\mathcal{E}} x$ with $\uparrow x' \subseteq \uparrow x$ to the arity inclusion function $\mathbf{inc}_{x',x} : \uparrow x' \hookrightarrow \uparrow x$. For any an entity type $x \in X$, the \mathcal{E} -tuple functor applied to the X -signature $\langle \uparrow x, \mathbf{inc}_x \rangle$ is the tuple set $\mathbf{tup}_{\mathcal{E}}(\uparrow x, \mathbf{inc}_x) = \{\uparrow x \xrightarrow{t} Y \mid t(x') \models_{\mathcal{E}} x', \forall x' \geq_{\mathcal{E}} x\}$. The key functor is the extent operator $\langle X, \leq_{\mathcal{E}} \rangle \xrightarrow{\mathbf{ext}_{\mathcal{E}}} \mathbf{Set}$, which maps an entity type $x \in X$ to its extent $\mathbf{ext}_{\mathcal{E}}(x) = \{y \in Y \mid y \models_{\mathcal{E}} x\}$ and maps the ordering $x \leq_{\mathcal{E}} x'$ with $\mathbf{ext}_{\mathcal{E}}(x) \subseteq \mathbf{ext}_{\mathcal{E}}(x')$ to the extent inclusion function $\mathbf{ext}_{\mathcal{E}}(x, x') : \mathbf{ext}_{\mathcal{E}}(x) \hookrightarrow \mathbf{ext}_{\mathcal{E}}(x')$. For any entity type $x \in X$, the



tional database schema $\mathcal{S} = \langle \mathbf{R}, X, \mathbf{S} \rangle$ with colimit reference schema $\langle I, s \rangle = \coprod \mathbf{S}$, defines a type language $\mathbf{lang}(\mathcal{S})$ in the Information Flow Framework [9] with reference component $\langle X, I, s \rangle$ and signature component $\langle \mathbf{R}, \partial \rangle$. We regard the colimit X -signature $\langle I, s \rangle = \coprod \mathbf{S}$ to be a reference schema $\langle X, I, s \rangle$ with reference (sort) function $I \xrightarrow{s} X$ from a universal set of variables I to the type set X . For any relation symbol $r \in \mathbf{R}$, the colimit injection $\mathbf{S}(r) = \langle I_r, s_r \rangle \xrightarrow{\iota_r} \langle I, s \rangle$, whose condition $s_r = \iota_r \cdot s$ expresses the s -alignment of s_r via ι_r , states that the signature $\langle I_r, s_r \rangle$ is below (at least as general as) the colimit signature $\langle I, s \rangle$. The signature functor ∂ factors $\mathbf{S} = \partial \circ \mathbf{inc} : \mathbf{R} \rightarrow (\mathbf{Set} \downarrow X)$ through $\mathbf{sign}(I, s) \subseteq (\mathbf{Set} \downarrow X)$, the subcategory of X -signatures below $\langle I, s \rangle$.

¹¹ Since any preorder $\mathcal{P} = \langle P, \leq \rangle$ is a classification $\mathcal{P} = \langle P, P, \leq \rangle$, a preorder is a relational database.

tuple function $\tau_{\mathcal{E}}(x) : \mathbf{ext}_{\mathcal{E}}(x) \rightarrow \mathbf{tup}_{\mathcal{E}}(\uparrow x, \mathbf{inc}_x)$ maps an entity instance $y \in \mathbf{ext}_{\mathcal{E}}(x)$ to the constant tuple $\uparrow x \xrightarrow{y} Y$. This defines a natural transformation $\tau_{\mathcal{E}} : \mathbf{ext}_{\mathcal{E}} \Rightarrow (\uparrow_{\mathcal{E}})^{\text{op}} \circ \mathbf{tup}_{\mathcal{E}}$.

IFF Structure. Using the key functor, we can define the relation classification $\mathcal{R} = \langle R, K, \models_{\mathcal{R}} \rangle$ with type set $R = \mathbf{obj}(\mathbf{R})$, instance set $K = \bigcup_{r \in \mathbf{R}} \mathbf{K}(r)$, and incidence with $k \models_{\mathcal{R}} r$ when $k \in \mathbf{K}(r)$ for key $k \in K = \bigcup_{r \in \mathbf{R}} \mathbf{K}(r)$ and relation symbol $r \in R = \mathbf{obj}(\mathbf{R})$. The elements (keys) in K are called abstract tuples in the Information Flow Framework [9]. Any relational database $\mathcal{D} = \langle \mathcal{S}, \mathcal{E}, \mathbf{K}, \tau \rangle$ determines an structure (model) $\mathbf{struc}(\mathcal{D})$ in the Information Flow Framework [9] with type language $\mathbf{lang}(\mathcal{S})$, entity classification \mathcal{E} , semidesignation $\langle I, s, \mathcal{E} \rangle$ and relation classification \mathcal{R} . This is an adjoint situation: any IFF structure determines a relational database.

Unified Database. A unified database is a special case of a database, whose relation classification coincides with its entity classification $\mathcal{R} = \mathcal{E}$. Unified databases allow the introduction of foreign keys. In fact, columns are either the single primary key or a foreign key. The entries in a column are keys of the type of the column. Actual datatypes, such as strings or numbers, can be regarded as primary keys of themselves. Conversely, we can think of any relational table with a single column, one whose schema is of the form $1 \xrightarrow{x} X$, to be a set of entities.

Any relational database schema $\mathcal{S} = \langle \mathbf{R}, X, \mathbf{S} \rangle$ in unified form ($R = \mathbf{obj}(\mathbf{R}) = X$) has an associated sketch. Define the arity functor $\mathbf{A} = \mathbf{S} \circ \mathbf{set}_X : \mathbf{R} \rightarrow (\mathbf{Set} \downarrow X) \rightarrow \mathbf{Set}$. Let $\int \mathbf{A} \xrightarrow{pr} \mathbf{R}$ denote the Grothendieck construction of \mathbf{A} with object set $\coprod_{r \in R} \mathbf{A}(r) = \{(r, i) \mid r \in R, i \in I, \langle I, s \rangle = \mathbf{S}(r)\}$ and morphisms $(r', i') \xrightarrow{p} (r, i)$ for \mathbf{R} -constraints $r' \xrightarrow{p} r$. The graph $\mathbf{gph}(\mathcal{S})$ of the sketch has node set R and edges $(r, i) \in \int \mathbf{A}$ with source and target $r \xrightarrow{(r, i)} s(i)$. This graph is actually 2-dimensional, given the \mathbf{R} -constraints. The sketch specifies a cone for the signature of each relation type $r \in \mathbf{R}$ and constraints for the commuting diagrams in \mathbf{R} . Any relational database $\mathcal{D} = \langle \mathcal{S}, \mathcal{E}, \mathbf{K}, \tau \rangle$ in unified form has an associated sketch interpretation $\mathbf{gph}(\mathcal{S})^{\text{op}} \rightarrow \mathbf{Set}$. The interpretation maps a node (relation type) $r \in \mathbf{R}$ to $\mathbf{K}(r)$ the set of keys of r and maps an edge $r \xrightarrow{(r, i)} s(i)$ to the map $\mathbf{K}(r) \rightarrow \mathbf{K}(s(i)) : k \mapsto \tau_r(k)(i)$, where $\tau_r(k) \in \mathbf{tup}_{\mathcal{E}}(I, s)$. This also is 2-dimensional.

A relational database morphism $\langle \mathbf{F}, \theta, f, g, \kappa \rangle : \mathcal{D}_2 = \langle \mathcal{S}_2, \mathcal{E}_2, \mathbf{K}_2, \tau_2 \rangle \rightarrow \langle \mathcal{S}_1, \mathcal{E}_1, \mathbf{K}_1, \tau_1 \rangle = \mathcal{D}_1$ consists of a relational database schema morphism $\langle \mathbf{F}, \theta, f \rangle : \mathcal{S}_2 \rightarrow \mathcal{S}_1$ ¹² an entity infomorphism $\langle f, g \rangle : \mathcal{E}_2 = \langle X_2, Y_2, \models_{\mathcal{E}_2} \rangle \rightleftarrows \langle X_1, Y_1, \models_{\mathcal{E}_1} \rangle = \mathcal{E}_1$ with a common (entity) type function component $f : X_2 \rightarrow X_1$ and a universe (entity instance) function $g : X_1 \rightarrow X_2$, and a key natural transformation

¹² A relational database schema morphism $\langle \mathbf{F}, \theta, f \rangle : \mathcal{S}_2 = \langle \mathbf{R}_2, X_2, \mathbf{S}_2 \rangle \rightarrow \langle \mathbf{R}_1, X_1, \mathbf{S}_1 \rangle = \mathcal{S}_1$ consists of a relation functor $\mathbf{F} : \mathbf{R}_2 \rightarrow \mathbf{R}_1$, a function on entity types $f : X_2 \rightarrow X_1$ and a signature natural transformation $\theta : \mathbf{S}_2 \circ \Sigma_f \Rightarrow \mathbf{F} \circ \mathbf{S}_1$. Any (strict) relational database schema morphism $\langle \mathbf{F}, f \rangle : \mathcal{S}_2 \rightarrow \mathcal{S}_1$ determines a type language morphism $\mathbf{lang}(\mathbf{F}, f) : \mathbf{lang}(\mathcal{S}_2) \rightarrow \mathbf{lang}(\mathcal{S}_1)$ in the Information Flow Framework [9], since we have the commutative diagram $\mathbf{F} \circ \partial_1 = \partial_2 \circ \mathbf{sign}(\coprod \mathbf{F}, f)$.

$\kappa : \mathbf{F}^{\text{op}} \circ \mathbf{K}_1 \Rightarrow \mathbf{K}_2$, which satisfy the condition $\kappa \bullet \tau_2 = \mathbf{F}^{\text{op}} \tau_1 \bullet \theta_{\langle f, g \rangle}^{\text{op}}$, where $\theta_{\langle f, g \rangle}^{\text{op}} \doteq \theta^{\text{op}} \mathbf{tup}_{\mathcal{E}_1} \bullet \mathbf{S}_2^{\text{op}} \tau_{\langle f, g \rangle}$.¹³ It is strict or trim when the underlying relational database schema morphism is strict or trim ($\theta = 1$). Figure 2 illustrates in detail a relational database morphism. Here are some examples of relational database morphisms.

Table morphism. A relational database morphism $\langle \mathbf{F}, \theta, f, g, \kappa \rangle$ with one-object source and target categories of relations and identity relation functor $\mathbf{F} = \mathbf{id}_1 : \mathbf{1} \rightarrow \mathbf{1}$ is identical to a single morphism of tables $\langle k, h, f, g \rangle : \langle K_1, t_1, I_1, s_1, \mathcal{E}_1 \rangle \rightarrow \langle K_2, t_2, I_2, s_2, \mathcal{E}_2 \rangle$, except that the direction has switched.

Infomorphism. An entity infomorphism $\langle f, g \rangle : \mathcal{E}_2 = \langle X_2, Y_2, \models_{\mathcal{E}_2} \rangle \rightleftarrows \langle X_1, Y_1, \models_{\mathcal{E}_1} \rangle = \mathcal{E}_1$ is a relational database morphism between the classifications regarded as relational databases, where the following hold.¹⁴ The type function is monotonic $f : \langle X_2, \geq_{\mathcal{E}_2} \rangle \rightarrow \langle X_1, \geq_{\mathcal{E}_1} \rangle$ mapping an ordering $x'_2 \geq_{\mathcal{E}_2} x_2$ with $\mathbf{ext}_{\mathcal{E}_2}(x'_2) \supseteq \mathbf{ext}_{\mathcal{E}_2}(x_2)$ to the ordering $f(x'_2) \geq_{\mathcal{E}_1} f(x_2)$, with $\mathbf{ext}_{\mathcal{E}_1}(f(x'_2)) \supseteq \mathbf{ext}_{\mathcal{E}_1}(f(x_2))$, since $y_1 \models_{\mathcal{E}_1} f(x_2)$ iff $g(y_1) \models_{\mathcal{E}_2} x_2$ implies $g(y_1) \models_{\mathcal{E}_2} x'_2$ iff $y_1 \models_{\mathcal{E}_1} f(x'_2)$. For each type $x_2 \in X_2$, the type function $X_2 \xrightarrow{f} X_1$ restricts to an arity function $\langle \uparrow_{\mathcal{E}_1}(x_2), \mathbf{inc}_{x_2} \cdot f \rangle \xrightarrow{f} \langle \uparrow_{\mathcal{E}_1}(f(x_2)), \mathbf{inc}_{f(x_2)} \rangle$. This is the x_2^{th} -component function of a signature natural transformation $\theta : \uparrow_{\mathcal{E}_2} \circ \Sigma_f \Rightarrow f \circ \uparrow_{\mathcal{E}_1}$. For each type $x_2 \in X_2$, the instance function $Y_1 \xrightarrow{g} Y_2$ restricts to a function $\mathbf{ext}_{\mathcal{E}_1}(f(x_2)) \xrightarrow{g} \mathbf{ext}_{\mathcal{E}_2}(x_2)$, since an instance $y_1 \in Y_1$ satisfying $y_1 \models_{\mathcal{E}_1} f(x_2)$ determines the instance $g(y_1) \in Y_2$ satisfying $g(y_1) \models_{\mathcal{E}_2} x_2$. This is the x_2^{th} -component function of a key natural transformation $g : \langle X_2, \leq_{\mathcal{E}_2} \rangle \xrightarrow{f^{\text{op}}} \langle X_1, \leq_{\mathcal{E}_1} \rangle \xrightarrow{\mathbf{ext}_{\mathcal{E}_1}} \mathbf{Set} \Rightarrow \langle X, \leq_{\mathcal{E}_2} \rangle \xrightarrow{\mathbf{ext}_{\mathcal{E}_2}} \mathbf{Set}$. Finally, for any entity type $x_2 \in X_2$, the two functions $\mathbf{ext}_{\mathcal{E}_1}(f(x_2)) \xrightarrow{g} \mathbf{ext}_{\mathcal{E}_2}(x_2) \xrightarrow{\Delta} \mathbf{tup}_{\mathcal{E}_2}(\uparrow_{x_2}, \mathbf{inc}_{x_2})$ and $\mathbf{ext}_{\mathcal{E}_1}(f(x_2)) \xrightarrow{\Delta} \mathbf{tup}_{\mathcal{E}_1}(\uparrow f(x_2), \mathbf{inc}_{f(x_2)}) \xrightarrow{f \cdot (-)} \mathbf{tup}_{\mathcal{E}_1}(\uparrow x_2, \mathbf{inc}_{x_2} \cdot f) \xrightarrow{(-) \cdot g} \mathbf{tup}_{\mathcal{E}_2}(\uparrow x_2, \mathbf{inc}_{x_2})$ are equal.

IFF Structure Morphism. Using the key natural transformation $\kappa : \mathbf{F}^{\text{op}} \circ \mathbf{K}_1 \Rightarrow \mathbf{K}_2$, we can define the relation infomorphism $\langle F, K \rangle : \mathcal{R}_2 = \langle R_2, K_2, \models_{\mathcal{R}_2} \rangle \rightleftarrows \langle R_1, K_1, \models_{\mathcal{R}_1} \rangle = \mathcal{R}_1$ with type function $F = \mathbf{obj}(\mathbf{F}) : R_2 \rightarrow R_1$ and instance function $K : K_1 \rightarrow K_2 : k_1 \mapsto \kappa_{r_2}(k_1)$ using the r_2^{th} component function $\kappa_{r_2} : \mathbf{K}_1(r_1) \rightarrow \mathbf{K}_2(r_2)$ for each key $k_1 \in \mathbf{K}_1(r_1)$ of an image relation type $r_1 = \mathbf{F}(r_2)$ (K is defined by arbitrary choice, otherwise). Any strict relational database morphism $\langle \mathbf{F}, f, g, \kappa \rangle : \mathcal{D}_2 = \langle \mathcal{S}_2, \mathcal{E}_2, \mathbf{K}_2, \tau_2 \rangle \rightarrow \langle \mathcal{S}_1, \mathcal{E}_1, \mathbf{K}_1, \tau_1 \rangle = \mathcal{D}_1$ determines an structure (model) morphism $\mathbf{struc}(\mathbf{F}, f, g, \kappa) : \mathbf{struc}(\mathcal{S}_2) \rightarrow \mathbf{struc}(\mathcal{S}_1)$ in the Information Flow Framework [9] with type language morphism $\mathbf{lang}(\mathbf{F}, f) : \mathbf{lang}(\mathcal{S}_2) \rightarrow \mathbf{lang}(\mathcal{S}_1)$, entity infomorphism $\langle f, g \rangle : \mathcal{E}_2 =$

¹³ $\tau_{\langle f, g \rangle}$ is the tuple natural transformation in the morphism of presheaves $\mathbf{tup}(f, g) = \langle \Sigma_f, f^*, \tau_{\langle f, g \rangle} \rangle : \langle (\mathbf{Set} \downarrow X_2), \mathbf{tup}_{\mathcal{E}_2} \rangle \rightleftarrows \langle (\mathbf{Set} \downarrow X_1), \mathbf{tup}_{\mathcal{E}_1} \rangle$ coming from the tuple functor $\mathbf{tup} : \mathbf{Cls} \rightarrow (\mathbf{Adj} \downarrow \mathbf{Set})$.

¹⁴ Since any pair of adjoint monotonic functions $\langle f, g \rangle : \langle P_2, \leq_2 \rangle \rightleftarrows \langle P_1, \leq_1 \rangle$ is an infomorphism $\langle f, g \rangle : \langle P_2, P_2, \leq_2 \rangle \rightleftarrows \langle P_1, P_1, \leq_1 \rangle$, such a pair is a relational database morphism.

$\langle X_2, Y_2, \models_{\mathcal{E}_2} \rangle \rightleftharpoons \langle X_1, Y_1, \models_{\mathcal{E}_1} \rangle = \mathcal{E}_1$, semidesignation morphism $\langle \coprod F, f, g \rangle : \langle I_2, s_2, \mathcal{E}_2 \rangle \rightarrow \langle I_1, s_1, \mathcal{E}_1 \rangle$ and relation infomorphism $\langle F, K \rangle : \mathcal{R}_2 \rightleftharpoons \mathcal{R}_1$.

Composition of morphisms is defined component-wise. Let **Db** denote the category of relational databases with the two projections (*dbs* is called the schema functor) $\mathbf{Dbs} \xleftarrow{dbs} \mathbf{Db} \xrightarrow{cls} \mathbf{Cls}$ and the key functor $\mathbf{Db} \xrightarrow{key} (\mathbf{Cat} \Downarrow \mathbf{Set})$ mapping \mathcal{D} to $\langle \mathbf{R}, \mathbf{K} \rangle$ and $\langle F, \theta, f, g, \kappa \rangle : \mathcal{D}_2 = \langle \mathcal{S}_2, \mathcal{E}_2, \mathbf{K}_2, \tau_2 \rangle \rightarrow \langle \mathcal{S}_1, \mathcal{E}_1, \mathbf{K}_1, \tau_1 \rangle = \mathcal{D}_1$ to $\langle F, \kappa \rangle : \langle \mathbf{R}_2, \mathbf{K}_2 \rangle \rightarrow \langle \mathbf{R}_1, \mathbf{K}_1 \rangle$.

$$\begin{array}{c}
 \langle \mathcal{S}_2, \mathcal{E}_2, \mathbf{K}_2, \tau_2 \rangle \xrightarrow{\langle F, \theta, f, g, \kappa \rangle} \langle \mathcal{S}_1, \mathcal{E}_1, \mathbf{K}_1, \tau_1 \rangle \\
 \langle \mathbf{R}_2, \mathbf{K}_2 \rangle \xrightarrow{\langle F, \kappa \rangle} \langle \mathbf{R}_1, \mathbf{K}_1 \rangle \quad (\mathbf{Cat} \Downarrow \mathbf{Set}) \xleftarrow{key} \mathbf{Db} \xrightarrow{cls} \mathbf{Cls} \quad \langle X_2, Y_2, \models_{\mathcal{E}_2} \rangle \xrightarrow{\langle f, g \rangle} \langle X_1, Y_1, \models_{\mathcal{E}_1} \rangle \\
 \begin{array}{ccc}
 \downarrow top & \swarrow sch & \nearrow SDsgn \\
 \mathbf{Cat} & \xleftarrow{rel} \mathbf{Dbs} & \xrightarrow{typ} \mathbf{Set} \\
 & \downarrow \langle F, \theta, f \rangle & \\
 & \langle \mathbf{R}_2, X_2, S_2 \rangle & \xrightarrow{\langle F, \theta, f \rangle} \langle \mathbf{R}_1, X_1, S_1 \rangle
 \end{array}
 \end{array}$$

Proposition 6. *There is a diagram functor $\mathbf{Db} \xrightarrow{dgm} (\mathbf{Cat} \Downarrow \mathbf{Tbl})$ from databases to (the lax comma category of) diagrams of tables.*

Recall that the limit operation is a functor $(\mathbf{Cat} \Downarrow \mathbf{Tbl})^{\text{op}} \xrightarrow{lim} \mathbf{Tbl}$.

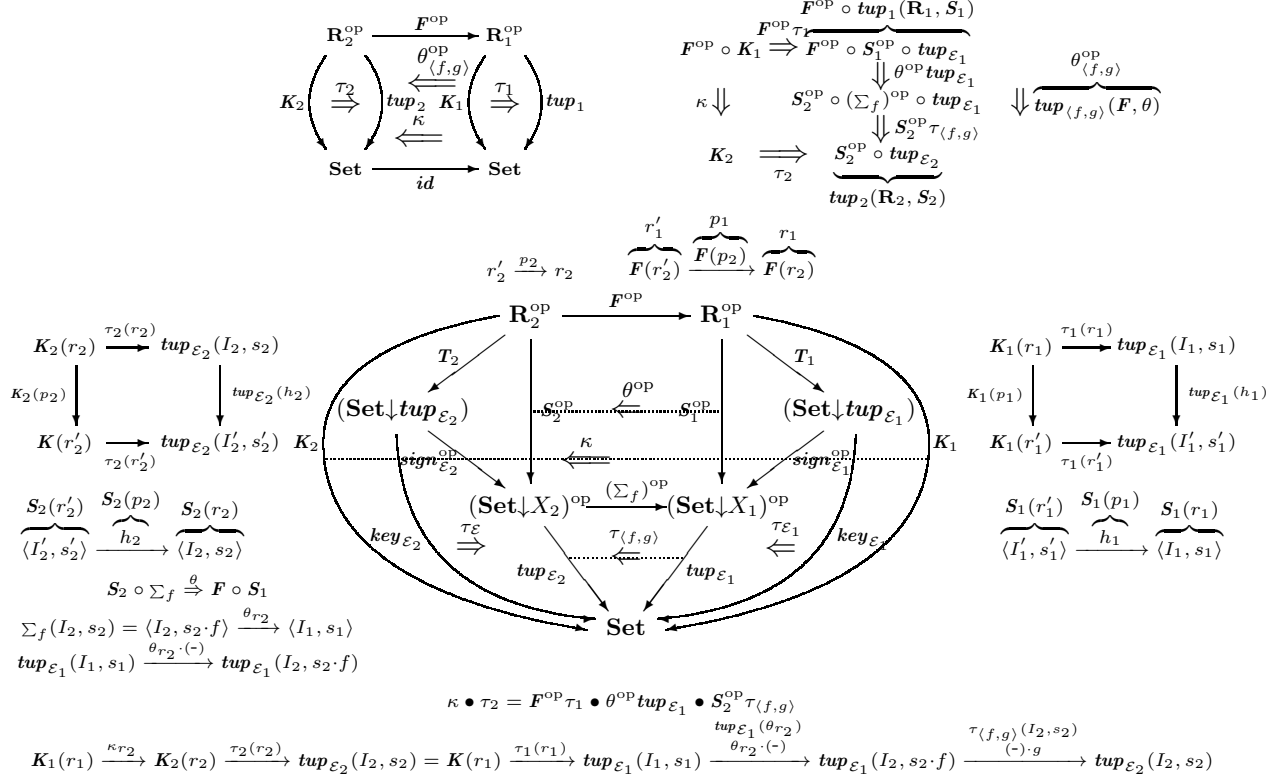
Definition 1. *The join functor is defined to be the composition*

$$join = dgm^{\text{op}} \circ lim : \mathbf{Db}^{\text{op}} \rightarrow \mathbf{Tbl}.$$

Corollary 1. *The schema of the join of a database is the reference (colimit) of the underlying database schema.*

$$\begin{array}{ccccc}
 & & (\mathbf{Cat} \Downarrow \mathbf{Tbl}) & & \\
 & dgm \nearrow & \downarrow join^{\text{op}} & \nwarrow lim^{\text{op}} & \\
 \mathbf{Db} & \xrightarrow{join^{\text{op}}} & \mathbf{Tbl}^{\text{op}} & & \\
 \downarrow dbs & \nearrow dgm & \downarrow sch & & \\
 \mathbf{Dbs} & \xrightarrow{refer} & \mathbf{Sch} & & \\
 & & \nwarrow colim & & \\
 & & (\mathbf{Cat} \Uparrow \mathbf{Sch}) & &
 \end{array}$$

In any complete category, the limits of arbitrary diagrams can be constructed by using only the terminal object and (binary) pullbacks. Dually, in any cocomplete category, the colimits of arbitrary diagrams can be constructed by using only the initial object and (binary) pushouts. As we have shown, for any entity classification $\mathcal{E} = \langle X, Y, \models_{\mathcal{E}} \rangle$, the category of \mathcal{E} -tables $\mathbf{Cat}(\mathcal{E})$ is complete. Hence, for any database schema \mathcal{S} the join of arbitrary \mathcal{S} -databases can be constructed by using only the join of the empty database (the terminal \mathcal{E} -table) and the join of \mathcal{E} -databases with binary span X -schemas (two \mathcal{E} -tables connected through a third).



This figure illustrates the condition on relational database morphisms. It has been annotated to help guide the understanding. The condition is symbolically stated in the two lines of text just above. The top line states the condition in terms of natural transformations. The bottom line states the condition in terms of set functions on the r_2^{th} component for some source relation type $r_2 \in \mathbf{R}_2$. The large diagram in the center illustrates the condition. The two upper diagrams give alternate views of this. The top right diagram is in a form very much like a table morphism. This is appropriate, since a relational database morphism between single table databases is just a table morphism. Finally, we have illustrated the effect of the morphism on the source/target tables, starting with a source relational constraint (morphism) $r'_2 \xrightarrow{p} r_2$.

Fig. 2. Relational Database Morphism

5 Summary and Future Work

We have defined the semantics for the relational database logical environment, which can be used to specify database system consequence. This provides interpretations for various formalisms such as relational algebra and first order logic, where terms and equations can be included by replacing signature morphisms with (possibly quotiented) term-tuples. The two most important achievements of this paper are the definition of a natural and general category of tables that is both complete and cocomplete, and the definition of a morphism of databases with some very nice properties. We have extended the notion of tables (Spivak [6]), first from an underlying entity type specification to an entity classification (models multi-inheritance), second from the static case of an underlying entity classification to the dynamic case of tables moving along an underlying entity infomorphism. We have proven completeness and cocompleteness for this (larger) category of tables. Completeness allows joins over arbitrary collections of tables that are possibly linked by projections. This includes selection, which is the join with respect to reference relations (tables). Cocompleteness allows a distributed union that is new.

However, much work needs to be done. We need to investigate further properties of database morphisms, including continuity. In a follow-up paper we will develop various formalisms, such as relational algebra and first order logic, and define views and queries. This will deepen the connection with the Information Flow Framework. Functional dependencies and normal forms should be expressed in terms of the categorical structure. For practical database maintenance, modifications (insertion, deletion and update) need to be defined. The unified form (plus its graphical representation) needs further development. And finally, the theory of databases defined in this paper should be more closely compared and contrasted with other approaches, such as the simplicial database approach (Spivak [6],[7]) and the sketch approach (Johnson, Rosebrugh et al [4]).

References

1. Barwise, J., Seligman, J.: Information Flow: The Logic of Distributed Systems. Cambridge University Press, Cambridge (1997)
2. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer, New York (1999)
3. Goguen, J., Burstall, R.: Institutions: Abstract Model Theory for Specification and Programming. J. Assoc. Comp. Mach. vol. 39, pp. 95–146, (1992)
4. Johnson, M., Rosebrugh, R.: Fibrations and Universal View Updatibility. Th. Comp. Sci. 388: 109–129 (2007).
5. Kent, R.E.: System Consequence. In: Rudolph, S., Dau, F., Kuznetsov, S.O. (eds.) LNCS vol. 5662, pp. 201–218 (2009). Springer, Heidelberg.
6. Spivak, D.I.: Simplicial Databases [<http://arxiv.org/abs/0904.2012>].
7. Spivak, D.I.: Functorial Data Migration [<http://arxiv.org/abs/1009.1166>].
8. Tarlecki, A., Burstall, R., Goguen, J.: Indexed Categories. Th. Comp. Sci. vol. 91, pp. 239–264. Elsevier (1991).
9. The Information Flow Framework (IFF) [<http://suo.ieee.org/IFF/>].